

Un livre de Wikilivres.

Programmation Python

Une version à jour et éditable de ce livre est disponible sur Wikilivres,
une bibliothèque de livres pédagogiques, à l'URL:
[//fr.wikibooks.org/wiki/Programmation_Python](https://fr.wikibooks.org/wiki/Programmation_Python)

Vous avez la permission de copier, distribuer et/ou modifier ce document
selon les termes de la Licence de documentation libre GNU, version 1.2 ou
plus récente publiée par la Free Software Foundation; sans sections
inaltérables, sans texte de première page de couverture et sans Texte de
dernière page de couverture. Une copie de cette licence est incluse dans
l'annexe nommée « Licence de documentation libre GNU ».

Définition du langage

Python est un langage de script de haut niveau, structuré et open source. Il est multi-paradigme et multi-usage.

Développé à l'origine par Guido Van Rossum en 1993, il est, comme la plupart des applications et outils open source, maintenu par une équipe de développeurs un peu partout dans le monde.

Conçu pour être orienté objet, il n'en dispose pas moins d'outils permettant de se livrer à la programmation fonctionnelle ou impérative; c'est d'ailleurs une des raisons qui lui vaut son appellation de « langage agile ».

Parmi les autres raisons, citons la rapidité de développement (qualité propre aux langages interprétés), la grande quantité de modules fournis dans la distribution de base ainsi que le nombre d'interfaces disponibles avec des bibliothèques écrites en C, C++ ou Fortran. Il est également apprécié pour la clarté de sa syntaxe, ce qui l'oppose au langage Perl.

Utilisation

Comme mentionné plus haut, Python se prête à un grand nombre de tâches. La distribution de base permet, entre autre, des développements réseau, la création d'interfaces graphiques (via tcl/tk), de la programmation cgi, de traiter du XML, etc... Sa relative facilité d'interfaçage avec des bibliothèques écrites en d'autres langages en fait un outil de choix pour des applications de calcul scientifique. Il est également de plus en plus utilisé comme langage de prototypage.

Python est aussi remarquable pour le nombre de bibliothèques accessibles via l'installation des modules appropriés. Que ce soit la connection avec une base de donnée, l'utilisation de bibliothèques d'interface graphique (wxPython, PyQt, pyGTK), la manipulation avancée de XML (pyXML), le traitement d'image (Python Imaging Library), le développement de jeu vidéo (pygame), OpenGL, la grande majorité des technologies actuelles dispose de son extension python.

Quelques exemples d'utilisation de Python

- Le serveur d'application Zope
- Administration du moteur de recherche Google
- Administration des fermes de rendu de la société d'effets spéciaux ILM
- l'application de dessin vectoriel Skencil (anciennement Sketch)
- Boa constructor, outil de développement rapide d'applications wxPython<

D'autres exemples sont disponibles sur Python success stories (anglais) (<http://pythonology.org/success>) [\[archive\]](#)

Plateformes

L'interpréteur Python est disponible sur de nombreux systèmes d'exploitation parmi lesquels on peut citer Microsoft Windows, Linux, Unix, Mac OS.

Installation

Il est possible d'installer Python sur les systèmes Microsoft ou sur d'autres systèmes Unix. Python est disponible ici : télécharger python (<http://www.python.org/download>) [\[archive\]](#).

Si vous n'avez pas les droits d'administrateurs de votre machine, il existe aussi une version portable (<http://www.portablepython.com>) [\[archive\]](#).

Pour les systèmes Windows, vous pouvez télécharger le fichier MSI. Mais, pour les systèmes Unix, Linux, et Mac OS, il faut compiler le programme à partir de fichiers source.

Python est le plus souvent automatiquement installé avec la plupart des distribution Linux ou Mac OS, mais il est recommandé (nécessaire) de télécharger une version actualisée.

Utilisation

Python dispose d'une interface interactive qui permet de tester les commandes de base. Pour appeler l'interface, il suffit de saisir *python* dans une console shell.

Vous pouvez également exécuter un script Python (fichier.py) en ligne de commande :

Exécuter un script python

```
> set PATH=%PATH%;C:\<Repertoire de python>
> python MonScript.py
```

Avec Windows, pour lancer un script en ligne de commande, il faut se rendre dans le répertoire où se situe le script et exécuter :

Exécuter un script python

```
> MonScript.py
```

Astuces pour démarrer

Lorsque vous souhaitez obtenir de l'aide sur une fonction ou une librairie, il suffit d'utiliser la commande « help » dans l'interpréteur interactif. Pour savoir comment fonctionnent les expressions régulières sous python par exemple, exécutez

Obtenir de l'aide sur une librairie python

```
> import re
> help(re)
```

Editeurs

- anglais Python scripter (<http://www.mmm-experts.com/Downloads.aspx>) [\[archive\]](#) : auto-complétion, navigation entre classe avec 'CTRL', génération et execution des tests unitaires, debugger ...
- Programmation Python/Python avec Eclipse



Éditeur par défaut

PyDev

anglais Pydev (<http://pydev.org/>) [\[archive\]](#) est un plug-in Eclipse pour le développement d'un projet Python (et Jython).

Il a été créé en Juillet 2003 par Aleks Totic et est maintenu depuis Octobre 2004 par Fabio Zadrozny. Il propose entre autres les fonctionnalités suivantes :

- complétion de code,
- analyse et mise en évidence de la syntaxe,
- debug

■ ...

Installation du plug-in PyDev

Sous Eclipse, voici la procédure à suivre pour installer le plug-in :

- Menu "Help" / "Software Updates" / "Find and install ..." / "Search for new feature to install"
- "New Remote Site..." / Name : `Python Dev`, URL : `http://pydev.org/updates/` / Finish

Une fois le plug-in installé, il faut configurer le compilateur Python :

- Menu "Window" / "Preferences" / "PyDev" + "Interpreter Python" / "New"

Il vous faudra ensuite choisir l'exécutable python : `"/usr/bin/python"` sous Linux, `"C:\Python\python.exe"` sous Windows et valider, puis sélectionner les répertoires à inclure dans le PYTHONPATH (en cas de doute, prenez ceux qui vous sont proposés).

Créer un projet "Hoo hoo World" avec le plug-in PyDev

Sous Eclipse, une fois le plug-in installé, choisir de créer un nouveau projet

- Menu "File" / "New" / "Project"
- Sélectionner `Pydev Project` / "Next"
- Donner un nom et choisir la version correspondant à python (ex: 2.4).
- Valider ("Finish") : *Vous devez avoir une nouvelle entrée correspondant au projet*
- Clic droit sur le projet pour ajouter un nouveau module ("New" / "Pydev module")
- donner lui un nom (ex: `monScript`)
- saisir le code suivant dans le fichier :

```
print "Hoo hoo World"
```

- sauvegarder (CTRL + S, ou clic sur la disquette, ou menu "File" / "Save")
- exécuter : Bouton droit sur le fichier `monScript` / "Run as" / "Python run"

Pour les prochaines exécutions du script, utiliser la barre d'outil (symbole lecture blanc dans un rond vert) ou CTRL + F11.

Complétion auto

Pour voir la complétion automatique de code, utilisez CTRL + Espace.

Par exemple pour le code suivant :

```
x = "Bonjour"
x.
```

si l'on place le curseur après le point, et que l'on tape CTRL + Espace, l'ensemble des méthodes de la classe String seront proposées (ex: `.upper()` qui passe en majuscule la chaîne de caractère `x`).

0. Installez le langage Python si vous êtes sous Windows, sinon il est déjà installé.

1. Ouvrir Python (command line). *Comment ?* En exécutant `python` ou en recherchant `python.exe`

Une invite de commande s'affiche

```
>>> _
```

Hello World !

Tradition, python2 : saisissez « `print "Hello World !"` » puis pressez Enter. Vous avez à l'écran :

```
>>> print "Hello World !"
Hello World !
>>> _
```

Python 3 : saisissez « `print("Bonjour tout le monde.")` » puis pressez Entrée/Retour. Vous avez à l'écran :

```
>>> print("Bonjour tout le monde.")
Bonjour tout le monde.
>>> _
```

2. Saisissez « `a=3` » puis pressez Entrée/Retour. Dans l'invite suivante saisissez « `print a` » (ou « `print(a)` » avec la version 3). Vous avez à l'écran :

```
>>> a=3
>>> print a
3
>>> _
```

3. Saisissez ces commandes (n'oubliez pas de modifier les *print* en rajoutant les parenthèses pour que le code soit compatible avec la version 3) :

```
>>> a=0                                # Shift+Enter pour sauter à la ligne suivante
>>> while (a<4):                       # ouvre un process (1), donc...
...     a = a+1 # ajouter une indentation (espace) au début pour signifier
...     print "semaine" , a           # demande d'afficher "semaine" et a.
...
semaine 1                             # 1er resultat
semaine 2
semaine 3
```

```
semaine 4                                     # etc, votre programme s'exécute en bouc
```

4. Ouvrez un éditeur de texte, écrivez...

```
a = 0
while (a<20):
    a = a + 1
    print "semaine" , a , "2009"
```

...et enregistrez ce script sous la forme *.py

5. Ouvrez la ligne de commande de votre système d'exploitation (par exemple : bash GNU/Linux, Terminal pour Mac OS X, ou cmd pour Windows), placez votre invite de commande dans le répertoire de votre fichier *.py, affichez votre fichier *.py dans votre invite de commande et exécutez-le en pressant Enter.

Félicitation, vous avez écrit et exécuté votre premier script !

Pour aller plus loin : Catégorie:Programmation.

... en deux minutes avec Python :

- une messagerie instantanée,
- un programme en interface avec Wikipédia,
- un serveur Web pour partager des fichiers et mettre en ligne un site statique en html.

Voir aussi le livre : Programmer en deux minutes

0. Pour programmer Version imprimable en deux minutes, nous allons écrire deux scripts en Python et les exécuter chacun dans une console. Les deux programmes vont communiquer afin d'envoyer un message et d'attendre la réponse.

Écoute

1. Ouvrir un éditeur de texte, et coller le script suivant (sans caractère spéciaux comme "é")...

ecoute.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
import socket          # importe un ensemble d'instructions pour connecter les programmes.
                        # Cet ensemble est disponible a l'installation de Python, dans la bibliotheque de base.

# Creation du connecteur d'ecoute par l'instruction 'socket'
# de la bibliotheque socket precedemment importee.
Connecteur = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

Hote = '127.0.0.1' # Adresse locale de l'ordinateur.
Port = 80         # Choix d'un port d'ecoute.
Connecteur.bind((Hote,Port)) # instruction 'bind' de la bibliotheque du connecteur
print "Le programme est a l'ecoute d'une eventuelle discussion, vous en serez averti." # Rajoutez des parent
Connecteur.listen(1)          # ecoute...
client, adresse = Connecteur.accept() # accepte...
print "L'ordinateur",adresse," veut discuter ! J'attends son message." # Rajoutez des parentheses pour Pytho
```

```
# Creation du connecteur de reponse
Reponse = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
Portreponse = 234
Reponse.connect((Hote,Portreponse ))
print "Note : je me suis connecte a",adresse," pour lui repondre" # Rajoutez des parentheses pour Python 3 !

while 1:
    Message = client.recv(255) # reception de la reponse, 255 caracteres max ; Python 3 : Message = str(
    if not Message:
        break
    print "\nMessage : ",Message,"\n" + "\n\nVotre reponse : " # Rajoutez des parentheses pour Python 3 !
    msgR = raw_input('>> ') # votre message ? Python 3 : msgR = bytes(input('>> '), 'mac_roman')
    Reponse.send(msgR) # envoi.

client.close() # ferme la connexion lorsque le client est parti : [ctrl+C] pour abandonner l'execution du pr
```

...enregistrez ce script (par exemple `ecoute.py`) et exécutez-le.

Discussion

2. Ouvrir l'éditeur de texte, écrire le script de discussion...

discussion.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
import socket
Discuter = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
Hote = '127.0.0.1'
Port = 80
Port_de_reponse = 234
Discuter.connect((Hote,Port)) # Se connecte au programme ecoute.py

Reponse = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
Reponse.bind((Hote,Port_de_reponse))
Reponse.listen(1)
client, adresse = Reponse.accept() # Creation du connecteur pour la reponse de ecoute.py
print "L'adresse",adresse," vous a entendu et attend votre message." # Rajoutez des parentheses pour Python
while 1:
    msg = raw_input('>> ') # votre message ? Python 3 : msg = bytes(input('>> '), 'mac_roman')
    Discuter.send(msg) # envoi.
    print "Attente de la reponse..." # Rajoutez des parentheses pour Python 3 !
    reponseaumessage = client.recv(255) # reception de la reponse, 255 caracteres max ; Python 3 : repon
    if not reponseaumessage:
        break
    print "\n",adresse,":",reponseaumessage,"\n" # affiche la reponse # Rajoutez des parentheses pour

client.close() # ferme la connexion lorsque le client quitte.
```

...enregistrez ce script (par exemple `discussion.py`) et exécutez-le dans une nouvelle console.

Félicitation, vos deux consoles communiquent !

0. Pour utiliser en deux minutes Version imprimable, nous allons écrire un script en Python et l'exécuter dans une console. Le script va utiliser deux ensembles de commandes définis dans la bibliothèque fournie à l'installation du langage.

L'interface avec Wikipédia se fait via requêtes HTTP ou HTTPS à wikipedia.org/w/api.php? ([//fr.wikipedia.org/w/api.php?](http://fr.wikipedia.org/w/api.php?)) [\[archive\]](#) - exemple ([//fr.wikipedia.org/w/api.php?action=query&prop=info%7Cvisions&titles=Accueil](http://fr.wikipedia.org/w/api.php?action=query&prop=info%7Cvisions&titles=Accueil)) [\[archive\]](#).

Réviseur de la page

1. Ouvrir un éditeur de texte, coller le script suivant (sans caractère spéciaux comme "é")...

reviseur_de_la_page.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
# Pour Python 3, remplacez la ligne ci-dessous par import urllib.request, re
import urllib, re # import des modules a partir de la bibliotheque d'instructions de base,
nom_de_page = "Accueil" # 'urllib' pour URL library et 're' pour regular expression.

url = "http://fr.wikipedia.org/w/api.php?action=query" +
      "&prop=info|revisions&titles=%s&format=xml" % nom_de_page
# affichage
# Python 3 : page = urllib.request.urlopen(url)
page = urllib.urlopen(url)
# Python 3 : infos = str(page.read(), 'utf_8')
infos = page.read() # lit le resultat de la requete a l'url ci-dessus
page.close()
print "Les informations demandees concernant",
      nom_de_page, "sont les suivantes, en XML :\n\n", infos # Rajoutez des parenthèses pour Python 3 !

# extraction
print "\n...recherche l'expression rationnelle..." # Rajoutez des parenthèses pour Python 3 !
reviseur= re.findall(' user="(.*?)" ',infos) # recherche l'expression rationnelle
print "\nDernier reviseur : ",reviseur # Rajoutez des parenthèses pour Python 3 !
```

...enregistrez ce script (par exemple `reviseur_de_la_page.py`) et exécutez-le. Le script utilise cette requête (<http://fr.wikipedia.org/w/api.php?action=query&prop=info%7Crevisions&titles=Accueil&format=xmlfm>) [\[archive\]](#) pour afficher le dernier réviseur de la page d'accueil.

Boucle réviseur bistro

2. Obtenir la liste des derniers réviseurs des Bistros du mois dernier. Ouvrir l'éditeur de texte, écrire ce script utilisant plusieurs fois cette requête (<http://fr.wikipedia.org/w/api.php?action=query&prop=info%7Crevisions&titles=Accueil>) [\[archive\]](#)... Si vous souhaitez utiliser le code suivant avec Python 3, faites les mêmes modifications que dans le script précédent. C'est-à-dire : rajoutez des parenthèses aux `print` ; chargez la classe `urllib.request` (au lieu d'`urllib` tout court) ; utilisez la fonction `urllib.request.urlopen` (au lieu de `urllib.urlopen`) ; transformez le résultat de `read` en chaîne de caractères (`infos = str(url.read(), 'utf_8')`).

boucle_reviseur_bistro.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
import urllib, re # import des modules de la bibliotheque d'instructions fournie a l'installation.

a=0
space = ' '
nospace = ''

while (a<31):
    a = a+1 # a est un nombre entier (integer)
    b = str(int(a)) # et b une chaine de caractere (string)
    nom = "Wikipedia:Le_Bistro/"+b+"_avril_2009"
    nom = nom.replace(space, nospace) # supprime les espace
    url = urllib.urlopen("http://fr.wikipedia.org/w/api.php?action=query" +
                          "&prop=info|revisions&titles=%s&format=xml" % nom )
```



```
infos = url.read()
url.close()
reviseur= re.findall(' user="(.*?)" ',infos) # recherche l'expression rationnelle
for chacun in reviseur:
    print "\nDernier reviseur du bistro du",b,"avril 2009 : ",chacun
```

...enregistrez ce script (par exemple `boucle_reviseur_bistro.py`) et exécutez-le.

Liste des réviseurs

3. La liste des réviseurs de la page d'accueil entre deux dates, et les commentaires de révisions : ouvrir l'éditeur de texte, écrire ce script, faire les mêmes modifications pour Python 3 le cas échéant... Ce script utilise cette requête ([//fr.wikipedia.org/w/api.php?action=query&prop=revisions&rvstart=20090311000000&revend=20090511000000&titles=Accueil](http://fr.wikipedia.org/w/api.php?action=query&prop=revisions&rvstart=20090311000000&revend=20090511000000&titles=Accueil)) [\[archive\]](#).

liste_des_reviseurs.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
import urllib, re # import des modules de la bibliotheque d'instructions fournie a l'installation.

debut = str(int(20090311000000)) # date pour commencer a lister, ici 11 mars 2009.
fin = str(int(20090511000000))

nom = "Accueil"

url = urllib.urlopen("http://fr.wikipedia.org/w/api.php?action=query" +
    "&prop=revisions&rvstart=%s&revend=%s&titles=%s&format=xml" % (debut, fin, nom))
infos = url.read()
url.close()

# recherche et affiche les reviseurs
#
reviseur= re.findall(' user="(.*?)" ',infos)
for chacun in reviseur:
    print "Reviseur : ",chacun

# recherche et affiche les commentaires
#
commentaire= re.findall(' comment="(.*?)" ',infos)
for comment in commentaire:
    print "\nCommentaire de revision : ",comment
```

Félicitations, vous utilisez Wikipédia via son **API** !

Vous pouvez poursuivre cet exercice en programmant du python sur une base vierge, ou alors utiliser la librairie d'instructions Pywikipedia et les scripts Pywikipedia hébergés par Wikimédia. Par exemple, vous devriez être capable de lire un script tel que :

statistics_in_wikitable.py (http://svn.wikimedia.org/viewvc/pywikipedia/trunk/pywikipedia/statistics_in_wikitable.py?view=markup)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
\03{lightyellow}This bot renders statistics provided by [[Special:Statistics]] in a table on a wiki page.\03
```

Thus it creates and updates a Statistics wikitable.

The following parameters are supported:

`\03{lightred}-screen\03{default}` If True, doesn't do any changes, but only shows the statistics.

`\03{lightgreen}-page\03{default}` On what page statistics are rendered.

If not existing yet, it is created.

If existing, it is updated.

"""

`__version__ = 'Id'`

`import wikipedia, pagegenerators, query`

`import time`

This is the title of the wikipage where to render stats.

`your_page = "Logstats"`

```
summary_update = {
    'en':u'Updating some statistics.',
}
```

```
summary_creation = {
    'en':u'Creating statistics log page.',
}
```

`class StatisticsBot:`

`def __init__ (self, screen, your_page):`

"""

Constructor. Parameter:

* `screen` - If True, doesn't do any real changes,
but only shows some stats.

"""

`self.screen = screen`

`self.your_page = your_page`

`self.dict = self.getdata() # Try to get data.`

`self.site = wikipedia.getSite()`

`def run(self):`

`if self.screen:`

`wikipedia.output("Bot is running to output stats.")`

`self.idle(1) # Run a function to idle`

`self.outputall()`

`if not self.screen:`

`self.outputall() # Output all datas on screen.`

`wikipedia.output("\nBot is running. " +`

`"Going to treat \03{lightpurple}%s\03{default}..." %`

`self.your_page)`

`self.idle(2)`

`self.treat()`

getdata() returns a dictionary of the query to

`api.php?action=query&meta=siteinfo&siprop=statistics`

`def getdata(self):`

This method return data in a dictionary format.

View data with: `api.php?action=query&meta=siteinfo&siprop=statistics&format=jsonfm`

`params = {`

`'action' : 'query',`

`'meta' : 'siteinfo',`

`'siprop' : 'statistics',`

`}`

`wikipedia.output("\nQuerying api for json-formatted data...")`

`try:`

`data = query.GetData(params,self.site, encodeTitle = False)`

`except:`

`url = self.site.protocol() + '://' + self.site.hostname() + self.site.api_address()`

`wikipedia.output("The query has failed. Have you check the API? Cookies are working?")`

`wikipedia.output(u"\n> \03{lightpurple}%s\03{default} <<" % url)`

`if data != None:`

`wikipedia.output("Extracting statistics...")`

`data = data['query'] # "query" entry of data.`

`dict = data['statistics'] # "statistics" entry of "query" dict.`

`return dict`

`def treat(self):`

`page = wikipedia.Page(self.site, self.your_page)`

`if page.exists():`

`wikipedia.output(u"\nWikitable on ' +`

`u'\03{lightpurple}%s\03{default} will be completed with:\n' % self.your_page)`

`text = page.get()`

`newtext = self.newraw()`

`wikipedia.output(newtext)`

`choice = wikipedia.inputChoice(`

```

        u'Do you want to add these on wikitable?', ['Yes', 'No'], ['y', 'N'], 'N')
    text = text[:-3] + newtext
    summ = wikipedia.translate(self.site, summary_update)
    if choice == 'y':
        try:
            page.put(u''.join(text), summ)
        except:
            wikipedia.output(u'Impossible to edit. It may be an edit conflict... Skipping...')
    else:
        wikipedia.output(
            u'\nWikitable on \03{lightpurple}%s\03{default} will be created with:\n' % self.your_page )
        newtext = self.newtable()+self.newraw()
        wikipedia.output(newtext)
        summ = wikipedia.translate(self.site, summary_creation)
        choice = wikipedia.inputChoice(
            u'Do you want to accept this page creation?', ['Yes', 'No'], ['y', 'N'], 'N')
        if choice == 'y':
            try:
                page.put(newtext, summ)
            except wikipedia.LockedPage:
                wikipedia.output(u"Page %s is locked; skipping." % title)
            except wikipedia.EditConflict:
                wikipedia.output(u'Skipping %s because of edit conflict' % title)
            except wikipedia.SpamfilterError, error:
                wikipedia.output(
                    u'Cannot change %s because of spam blacklist entry %s' % (title, error.url))

def newraw(self):
    newtext = ('\n|----\n!\'\'+ self.date() +'\'\')    # new raw for date and stats
    for name in self.dict:
        newtext += '\n|'+str(abs(self.dict[name]))
    newtext += '\n|----\n|}'
    return newtext

def newtable(self):
    newtext = ('\n{| class=wikitable style=text-align:center\n!' + "date")    # create table
    for name in self.dict:
        newtext += '\n|'+name
    return newtext

def date(self):
    return time.strftime('%Y/%m/%d', time.localtime(time.time()))

def outputall(self):
    list = self.dict.keys()
    list.sort()
    for name in self.dict:
        wikipedia.output("There are "+str(self.dict[name])+" "+name)

def idle(self, retry_idle_time):
    time.sleep(retry_idle_time)
    wikipedia.output(u"Starting in %i second..." % retry_idle_time)
    time.sleep(retry_idle_time)

def main(your_page):
    screen = False # If True it would not edit the wiki, only output statistics
    _page = None


    wikipedia.output("\nBuilding the bot...")
    for arg in wikipedia.handleArgs():    # Parse command line arguments
        if arg.startswith('-page'):
            if len(arg) == 5:
                _page = wikipedia.input(u'On what page do you want to add statistics?')
            else:
                _page = arg[6:]
        if arg.startswith("-screen"):
            screen = True
    if not _page:
        _page = your_page
    if not screen:
        wikipedia.output("The bot will add statistics on %s.\n" % _page )
    bot = StatisticsBot(screen, _page) # Launch the instance of a StatisticsBot
    bot.run() # Execute the 'run' method

if __name__ == "__main__":
    try:
        main(your_page)
    finally:
        wikipedia.stopme()

```

Le script `statistics_in_wikitable.py` importe quelques librairies d'instructions dont Pywikipedia, définit trois variables, définit l'objet `StatisticsBot`, puis définit une fonction principale qui est exécutée à la fin du script (par l'instruction `try: main(your_page)`).





Voir aussi


- Catégorie:Bot_publié_en_python sur Wikipédia 

Ce chapitre permet d'aborder les concepts fondamentaux du langage Python : les instructions, les blocs, les variables et leur type et enfin les opérateurs. Dans ce chapitre, les types et les opérateurs sont juste abordés de manière générale; ils sont détaillés pour les types très communs (nombre et booléen). Les types complexes (fichier et chaîne de caractères) et composés (tuple, liste et dictionnaire) seront abordés dans des sections spécifiques du chapitre *type de données complexe*.

Mis a part la manière de traiter le bloc d'instructions, le langage Python ne possède pas de particularités comparés aux autres langages héritiers de l'Algol (C, Java, PHP, etc.)

Ce chapitre est découpé en quatre parties :

1. Structure d'un programme 
2. Variable 
3. Type 
4. Opérateur 

- Exercices sur les bases du langage 

La structure d'un programme Python est certainement ce qui étonne le plus le programmeur ayant l'habitude d'un langage plus traditionnel comme le C. L'absence de point-virgule (;) et surtout l'absence d'accolade ({}) sont en effet déroutantes à première vue.

Les instructions

Un programme Python est composé d'**instructions**. Une instruction est un ordre unitaire donné à un programme. Par exemple *afficher Bonjour* est une instruction, de même que *calculer un plus un*.

Les instructions sont séparées dans le programme, soit par un point-virgule (;), soit par un retour à la ligne.

L'interpréteur Python commence par analyser la première ligne :

- si celle-ci contient une instruction, alors il l'**exécute**
- si l'instruction n'est pas une **instruction de contrôle**, alors, il passe à la ligne suivante, l'analyse et l'exécute
- si le programme Python arrive à la fin du fichier à exécuter, alors, il **sort** du programme et en arrête l'exécution.

```
print "Bonjour"
1+1
```

```
c = 3e5
e = m*c**2
a, b, c = 1, 2, 3
discriminant = b**2-4*a*c
lambda x,y : x + y
dicol = {'prénom':'Eric', 'nom':'tartempion'}
print ("Bonjour %s" % dicol['nom'])
```

Exemple 1 : quelques instructions

Les commentaires

Une ligne commençant par un dièse (#) n'est pas prise en compte par l'interpréteur. De même pour tous les caractères qui sont mis à la suite d'un dièse. Cette dernière règle ne s'applique pas lorsque le dièse est positionné dans une chaîne de caractères.

```
#Toute cette ligne est un commentaire.
print "Bonjour le monde" #Ceci est également un commentaire
print "Bonjour"; print "Le monde"; #Ceci est une ligne comportant
#plusieurs instructions
print "Cette ligne ne contient pas de #commentaire"
```

Exemple 1 : On notera que la fonction print affiche son argument.

Notion de bloc d'instructions

Un **bloc d'instructions** est une suite d'instructions qui est alignée sur la même tabulation. Les blocs d'instructions sont créés par les instructions de contrôles comme *if*, *while* et *for*, ainsi que par les instructions permettant de déclarer des **fonctions**.

```
#Ce bloc d'instruction est collé contre le bord gauche du fichier
print "Je suis dans le premier bloc"
print "Je suis toujours dans le premier bloc"

if (a == 12) : #L'instruction 'if' initie un nouveau bloc
    #Ce bloc est a quatre espace du bord
    print "Je suis dans le second bloc"
    print "Je suis encore dans le second bloc"

    if (b == 13 ) :
        #Il est possible d'imbriquer des blocs dans des blocs
        print "Je suis dans un troisième bloc"
        print "et ici aussi"

    print "Je reviens dans le second bloc"

print "Je suis revenue dans le premier bloc"
```

Exemple 2 : Les blocs

Il est très important de noter que les blocs doivent être *tracés au cordeau*. C'est à dire que si j'initie mon bloc avec huit espaces avant la première instruction, je dois toujours mettre huit espaces devant chacune des

instructions de ce bloc. Il n'est pas possible de mélanger des tabulations et des espaces. Si deux lignes semblent se trouver à la même distance du bord gauche, mais que l'une contient des espaces et l'autre des tabulations, alors l'interpréteur pourra générer l'erreur suivante :

IndentationError: unindent does not match any outer indentation level

Pour éviter ce genre d'écueil, il convient de toujours utiliser le même type d'indentation. Vous pouvez, par exemple, utiliser la touche tabulation et configurer votre éditeur pour qu'il remplace la tabulation par des espaces, ou utiliser de vraies tabulations. La première méthode est notamment la convention utilisée pour la librairie standard de Python, la seconde ayant l'avantage de fonctionner avec tous les éditeurs (et ainsi, de pouvoir être plusieurs sur le même projet avec des éditeurs différents).

Une **variable** est un espace mémoire dans lequel il est possible de mettre une **valeur**. Par exemple, si en français je dis *x est égal à 1*, j'utilise la variable dont le nom est *x* pour lui fixer la valeur *1*. Pour faire la même chose en Python, je note simplement :

Une simple affectation

```
x=1
```

On dit que « je procède à l'**affectation** de la variable *x* ». En Python, le symbole `=` est également nommé l'**opérateur d'affectation**.

Création de variables

En Python, les variables sont créées automatiquement à leur première utilisation. Pour créer une variable, il suffit donc de l'utiliser en l'affectant une première fois, c'est à dire d'écrire *nom_variable=valeur_de_la_variable*.

La valeur de la variable peut être un **littéral**, c'est à dire une constante, ou bien une **expression**. L'expression est évaluée avant d'être affectée à la variable

création de variables

```
x = 1 #On affecte la variable x avec la valeur du littéral 1
y = 1+1 #On affecte la variable y avec la valeur de l'expression 1 + 1
z = x + y
```

Nom des variables

2.x

Une variable peut prendre n'importe quel nom, tant qu'elle respecte les règles suivantes :

- Son nom commence par une lettre minuscule (a à z) ou majuscule (A à Z), ou bien par le caractère *souligné* (`_`)

- Pour la suite de son nom, on peut utiliser les lettres minuscules et majuscule, le souligné et un chiffre (0 à 9)
- Son nom ne doit pas être un **mot réservé**.

3.x

Les règles ont été assouplies, il devient possible d'utiliser n'importe quel caractère unicode non opérateur dans les noms, il demeure impossible de faire commencer le nom de variable par un chiffre ou d'utiliser des mots réservés.

Les noms suivants peuvent être utilisés comme nom de variable

```
x
X
toto
Toto
_
_toto
_toto_
```

Il faut également noter que les variables dont le nom commence par le caractère `_` ont une signification particulière :

- les noms commençant par un `_` ne sont pas exportés lorsqu'ils se trouvent dans un module;
- les noms commençant par deux `_` et finissant par deux `_` sont réservés par le langage lui même, notamment pour la Programmation orienté objet.

Le nom des variables est sensible à la casse, ainsi *toto* et *Toto* ne désignent pas la même variable.

Les noms suivants ne peuvent pas être utilisés comme nom de variable

```
41toto #On ne commence pas par un chiffre
éric #Contient un caractère ''é'' qui n'est pas autorisé
```

Affectations

- simples

```
x=7
```

ici `x = 7`

- multiples

```
x=y=7
```

ici `x` et `y = 7`

- parallèles

```
x, y = 7, 8
```

ici x=7 et y=8

Conseil sur le nom des variables

En général, les noms de variables sont toujours en minuscule, la majuscule du début est souvent réservée pour les noms des classes. Pour nommer une variable il convient surtout de la nommer en fonction de ce qu'elle est censée faire ou représenter.

Ainsi, *x*, *toto*, *All*, ne sont pas de bons noms de variable. Par contre *hauteur*, *personne*, *cellule* désignent mieux ce qu'elles sont censées représenter et aident un autre programmeur à comprendre ce qui a été fait.

Si le nom d'une variable doit comporter plusieurs mots, il y a deux possibilités d'écrire le nom de la variable :

- à la *C*, c'est à dire en séparant les mots par le caractère `_` Exemple : *marge_brut*
- à la *Java*, c'est à dire en séparant les mots par un passage en haut de casse (lettre majuscule). Exemple : *margeBrut*

Il est mieux d'éviter les abréviations : nb pour nombre, h au lieu de hauteur, etc. Il convient également d'éviter autant que possible l'énumération de variables (*toto1*, *toto2*, *toto3*, ...), cela rend le programme parfaitement incompréhensible et sujet à des erreurs.

Il est possible de préfixer le nom de la variable par son type. Par exemple *int_margeBrut*, *str_message_de_bienvenue*. Cela alourdit très fortement le programme. On pourra par exemple s'inspirer de la notation hongroise qui formalise ce mécanisme.

D'autres notations existent ou peuvent être imposées en fonction d'un projet, des habitudes d'une entreprise, etc.

Les références

Formellement, les variables Python sont des **références**, c'est à dire que écrire *a = ((1,0,0),(0,1,0),(0,0,1))* ne signifie pas que *a* vaut *((1,0,0),(0,1,0),(0,0,1))* mais que *a* référence le n-uplet *((1,0,0),(0,1,0),(0,0,1))*. La différence est que ensuite, une autre variable peut référencer le même n-uplet, simplement avec *b = a*. Si on modifie *b*, alors *a* sera également modifié.

L'utilisation des références

```
>>> a = [(1,0,0), (0,1,0), (0,0,1)]
>>> b = a
>>> print a
[(1,0,0), (0,1,0), (0,0,1)]
>>> print b
[(1,0,0), (0,1,0), (0,0,1)]
>>> b[0] = (1,1,0)
>>> print b
[(1,1,0), (0,1,0), (0,0,1)]
>>> print a
[(1,1,0), (0,1,0), (0,0,1)]
```

Notez que *a* et *b* ne sont pas "liés", ils référencent simplement le même objet.

Si ensuite on fait référencer à *a* un nouvel objet (par l'intermédiaire de l'opérateur d'affectation "="), *b*

référencera toujours l'ancien objet, et une modification de l'un des objets ne modifiera pas l'autre.

Indépendances des variables

```
>>> a = [1,2,3]
>>> b = a
>>> print a
[1,2,3]
>>> print b
[1,2,3]
>>> a = [4,5,6]
>>> print b
[1,2,3]
>>> print a
[4,5,6]
>>> a.append(7)
>>> print a
[4,5,6,7]
>>> print b
[1,2,3]
```

Idem pour les nombres :

Références et nombres

```
>>> a = 5
>>> b = a
>>> print a
5
>>> print b
5
>>> b += 5
>>> print b
10
>>> print a
5
```

L'objectif de cette section est la découverte des types de données Python et de la façon dont on les écrit sous forme littérale. L'étude complète des types complexes nécessite l'étude complète des structures de contrôle et sera donc abordée postérieurement.

Type de données

Python est un langage dont le typage est automatique. Cela signifie que bien que gérant différents types, lorsqu'une variable est affectée, l'interpréteur trouvera automatiquement son type.

Liste des types	
<i>int</i>	Nombre entier optimisé
<i>long</i>	Nombre entier de taille arbitraire
<i>float</i>	Nombre à virgule flottante
<i>complex</i>	Nombre complexe
<i>str</i>	Chaîne de caractère

<i>unicode</i>	Chaîne de caractère unicode
<i>tuple</i>	Liste de longueur fixe
<i>list</i>	Liste de longueur variable
<i>dict</i>	dictionnaire
<i>file</i>	Fichier
<i>bool</i>	Booléen
<i>NoneType</i>	Absence de type
<i>NotImplementedType</i>	Absence d'implémentation
<i>function</i>	fonction
<i>module</i>	module

Tableau 2 : Liste des types prédéfinis en Python

Tableau des types Python 3.x^[1]

nom	description	équivalent 2.x
int	nombre entier de longueur arbitraire	long
byte	chaîne de caractères ASCII	str
str	chaîne de caractères Unicode	unicode

La fonction *type()*

La fonction `type()` permet de connaître le type d'une variable

```
>>> a=3
>>> type(a)
<type 'int'>
```

Exemple 1 : utilisation de la fonction type

Les nombres

Il existe deux types pour définir des nombres entiers : le type **int** et le type **long**. Il existe également un type pour représenter des nombres à virgule : le type **float**.

Les nombres entiers de type *int*

Les int représentent le type le plus facilement représentable sur une architecture donnée. Par exemple, sur une machine 32-bits, la taille d'un int sera de 32-bit, donc un int permettra de représenter des nombres entre $-2^{31} + 1$ et $2^{31} - 1$, soit entre -2 147 483 647 et 2 147 483 647.

Un littéral int s'écrit tout simplement avec les chiffres de 0 à 9, précédé éventuellement du symbole -. Il est possible d'écrire ce littéral suivant trois bases :

- la base décimale : le littéral devra commencer par un chiffre entre 1 et 9

Un nombre à virgule flottante est un nombre décimal qu'il est possible de représenter par sa mantisse et son exposant. Par exemple, le nombre 125,789 est représentable par le couple (mantisse = 1,25789, exposant = 2). La mantisse étant toujours comprise entre -10 et 10 exclus.

Les nombres sont traduits par la formule *$nombre = mantisse * 10^{exposant}$* .

Les limites dépendent de l'architecture de la machine et sont équivalentes au type de donnée double du langage C.

Les littéraux peuvent s'écrire avec les chiffres, le caractère point pour indiquer la séparation entre la partie entière et la partie décimale et la lettre 'e' ou 'E' pour spécifier l'exposant.

```
x = 1.234
x = 1.0 #Notons qu'un entier peut être un flottant
x = 1. #Même résultat que précédemment
x = 1.234e54 #C'est à dire  $1.234 * 10^{54}$ 
x = 1.234E54 #idem
x = -1.454e-2 #La mantisse et l'exposant peuvent être négatifs
```

Exemple 5 : nombres à virgules

Les nombres complexes

Python est un des rares langages à proposer un type de base pour les nombres complexes \mathbb{C} . Un nombre complexe est un nombre composé d'une partie réelle et d'une partie imaginaire. On note i une des racines du polynôme $x^2 = -1$.

(voir aussi l'article de wikipédia sur les nombres complexes)

Dans certains contextes, (comme la physique), la racine de -1 est notée j. C'est ce qui se fait en Python.

Un littéral complexe s'écrit donc : a + bj, avec a et b des variables de type float. Attention, j doit être précédé d'un nombre car sinon, Python l'interprétera comme étant la variable j. Il ne doit pas y avoir d'espace entre ce nombre et j.

```
x = 1 + 1j
x = 1.2e3 + 1.5e7j
x = 5j + 4
x = 1 + x*1j
```

Exemple 5 : quelques nombres complexes

Les booléens

Un booléen est un type de données qui ne peut prendre que deux valeurs : vrai ou faux. En Python, les constantes littérales sont notées *True* et *False*.

Tous les types de variables peuvent être interprétés de manière booléenne. Par exemple, pour les entiers (int),

la valeur 0 correspond à faux et les autres valeurs à vrai. Il en est de même pour tous les autres types : une valeur particulière vaut *False* et le reste des valeurs *True*. Le tableau suivant présente les valeurs "faux" pour les principaux type de données.

Les valeurs False	
bool	False
int	0
float	0.
string	" "
tuple	()
list	[]
dict	{ }

Tableau 3 : Liste des valeurs évaluées à False

Les chaînes, introduction

Les chaînes de caractères sont des suites de caractères non modifiables en Python. Les littéraux *chaînes* s'écrivent soit entre guillemets, soit entre simple quote. Il est possible de mettre des quote (respectivement des guillemets) dans une chaîne entouré de guillemets (respectivement, de quote).

```
a = "Bonjour"
a = 'Bonjour'
a = "Je m'appelle Pierre"
a = 'Il lui a dit : "Bonjour" '
```

Exemple 7 : chaînes de caractère

Pour une étude plus approfondie sur les chaînes, voir la section sur les chaînes de caractères.

La conversion des types

Il existe plusieurs fonctions qui permettent de forcer le type d'une variable en un autre type.

- `int()` : permet de modifier une variable en entier. Provoque une erreur si cela n'est pas possible.
- `long()` : transforme une valeur en long.
- La fonction `str()` permet de transformer la plupart des variables d'un autre type en chaînes de caractère.
- `float()` : permet la transformation en flottant.
- `repr()` : similaire à `str`. Voir la partie sur les objets
- `eval()` : évalue le contenu de son argument comme si c'était du code Python.

Références

1. seuls les types ayant changé de nom ou de valeur sont mentionnés

Voir aussi

- Programmation Python/Type de données complexe

L'objectif de cette page est de permettre la découverte des différents opérateurs. L'utilisation des opérateurs sur des types complexes est reportée après l'étude des structures de contrôles.

Définition

Un *opérateur* est un symbole utilisé pour effectuer un calcul entre des **opérandes**.

Une opérande est une variable ou un littéral ou bien une **expression**.

Une expression est une suite valide d'opérateurs et d'opérandes.

Par exemple, dans l'expression :

```
x = y + 1
```

Il y a deux opérateurs (= et +) et trois opérandes (x,y et 1).

Il existe différents types d'opérateur :

- les opérateurs logiques
- les opérateurs de comparaisons
- les opérateurs sur les séquences
- les opérateurs numériques
- les opérateurs d'affectation

Les opérateurs peuvent avoir des sens différents en fonction des types d'opérandes sur lesquels ils agissent.

Présentation des différents opérateurs

les opérateurs logiques

Les expressions avec un opérateur logique sont évaluées à True ou False

- **X or Y** : OU logique, si X évalué à True, alors l'expression est True et Y n'est pas évalué. Sinon, l'expression est évaluée à la valeur booléenne de Y.
- **X and Y** : ET logique, si X est évalué à False, alors l'expression est False et Y n'est pas évalué. Sinon, l'expression est évaluée à la valeur booléenne de Y.
- **not X** : évalué à la valeur booléenne opposée de X.

les opérateurs de comparaisons

Tout comme les opérateurs logiques, les opérateurs de comparaison renvoient une valeur booléenne True ou False. Les opérateurs de comparaisons s'appliquent sur tous les types de base.

- < strictement inférieur
- > strictement supérieur

- `<=` inférieur ou égal
- `>=` supérieur ou égal
- `==` égal
- `!=` différent
- `<>` différent, on utilisera de préférence `!=`
- `X is Y` : X et Y représentent le même objet.
- `X is not Y` : X et Y ne représentent pas le même objet

Il est possible d'enchaîner les opérateurs : `X < Y < Z`, dans ce cas, c'est Y qui est pris en compte pour la comparaison avec Z et non pas l'évaluation de `(X < Y)` comme on pourrait s'y attendre dans d'autres langages.

les opérateurs mathématiques

symbole	effet	exemple
+	addition	<code>6+4 == 10</code>
-	soustraction	<code>6-4 == 2</code>
*	multiplication	<code>6*4 == 24</code>
/	division	<code>6/4 == 1.5</code>
**	élévation à la puissance	<code>12**2 == 144</code>
//	division entière	<code>6//4 == 1</code>
%	reste de la division entière	<code>6%4 == 2</code>

les opérateurs d'affectation

- `'='`
- Affectation multiple, aka `x = y = z = 3`
- Affectation parallèle aka `x,y = 1, 0.5`

Priorité des opérateurs

- explication
- tableau de priorité des opérateurs

Quelques fonctions

- `print` : pour afficher du texte à l'écran. Exemple : `print"texte"` # dans la branche 3.x de Python, il est nécessaire d'écrire: `print ("texte")`
- `raw_input` : insère dans une variable ce que l'utilisateur a rentré. Exemple : `variable = raw_input("il est possible d'afficher du texte ici")`

Exercices

Exercice 1

Écrire un programme qui affiche "Bonjour le monde".

Solution

```
print ("Bonjour le monde")
```

Exercice 2

Écrire un programme qui permet de saisir le nom de l'utilisateur et de renvoyer "Bonjour", suivi de ce nom

Solution

```
nom = raw_input("Quel est votre nom ? ")
print "Bonjour " + nom
```

ou `print "Bonjour " + raw_input("Quel est votre nom ? ")`

En version 3.1 cela donne

```
def Bonjour(name):
    print("Bonjour " + (name) + "!")
print("Quel est votre nom?")
name = input()
Bonjour(name)
```

Exercice 3

3. Écrire un programme qui demande à l'utilisateur la saisie de a et b et affiche la somme de a et de b.

Solution

```
a = raw_input("Valeur de a :")
b = raw_input("Valeur de b :")
print "Somme de a+b = " + str(int(a)+int(b))
```

En version 3.1 cela pourrait être

```
def Valeurs(var_a,var_b):
    print("Voici le résultat de a + b :"+str(var_c)+"!")
print("Donnez moi la valeur de a")
var_a = input ()
print("Donnez moi la valeur de b")
```



```
var_b = input ()
var_c=int(var_a)+int(var_b)
Valeurs(var_a,var_b)
```

Exercice 4

4. Écrire un programme qui demande à l'utilisateur son année de naissance et qui affiche son âge. L'année courante sera mise dans une variable.

Solution

```
# Ce script ne gère pas si l'anniversaire est passé ou non d'où le "environ"
annee_courante = 2013
print "Quelle est votre année de naissance ?"
reponse = raw_input()
print "Vous avez environ", annee_courante- int(reponse), "ans"
```

Pour Python 3.1

```
# Ce script ne gère pas si l'anniversaire est passé ou non
def Age(nom,age):
    print("Bonjour "+(nom)+" , vous avez "+str(age)+" ans!")
print("Quel est votre nom?")
nom=input ()
print("En quelle année êtes vous né ?")
annee=input()
annee_courante=2013
age=int(annee_courante)-int(annee)
Age(nom,age)
```

Exercice 5

5. Écrire un programme qui demande à l'utilisateur les coordonnées de deux points dans le plan et qui calcule puis affiche la distance entre ces deux points selon la formule : $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Solution

```
import math
print "xA?"
```

```
xA = int(raw_input())
print "yA?"
yA = int(raw_input())
print "xB?"
xB = int(raw_input())
print "yB?"
yB = int(raw_input())
print "distance entre A et B:", math.sqrt( (xA-xB)**2 + (yA-yB)**2)
```

En version 3.1

```
import math
def Distance(xA,yA,xB,yB):
    print("Bonjour, la distance qui sépare les deux points est de "+str(dista
print("Donnez moi Xa")
xA=int(input ())
print("Donnez moi Ya")
yA=int(input ())
print("Donnez moi Xb")
xB=int(input ())
print("Donnez moi Yb")
yB=int(input ())
distance=math.sqrt (( (xB-xA)**2)+( (yB-yA)**2) )
Distance(xA,yA,xB,yB)
```

notion de bloc, instruction pass

De manière générale, un bloc contient tout le code avec une même indentation.

```
ceci est le bloc principal
if condition:
    bloc 2
    if condition2:
        bloc 3
    fin du bloc 2
fin du bloc 1
```

Si à un endroit on a syntaxiquement besoin d'un bloc mais qu'il n'y a rien à faire, on peut utiliser l'instruction `pass`, qui justement ne fait rien

```
if condition:
    pass
else:
    instruction ...
```

l'instruction if

On utilise très souvent cette structure de contrôle. Sa syntaxe est très simple :

```
if condition:
    instructions
    ...
```

L'indentation après le ":" est obligatoire.

Cette structure de contrôle permet de tester une condition et de n'exécuter les instructions que si cette condition est vérifiée. Exemple :

```
a = 11
if a > 10 :
    print "a est plus grand que dix"
```

"if" veut dire "si" en français.

En exécutant ce programme, on voit "a est plus grand que dix" apparaître à l'écran. On peut perfectionner le programme pour prendre en compte le cas où a est plus petit que dix :

```
if a > 10 :
    print "a est plus grand que dix"
else:
    print "a n'est pas plus grand que dix"
```

"else" veut dire "sinon" en français.

On utilise aussi parfois elif (contraction de "else if") :

```
if a > 10 :
    print "a est plus grand que dix"
elif a == 10:
    print "a est égal à dix"
else:
    print "a est plus petit que dix"
```

l'instruction while

permet d'exécuter des commandes **tant qu'**une ou plusieurs conditions sont vraies.

```
while condition:
    commandes
```

par exemple :

```
i=0
while i<5:
    i=i+1
    print i,
```

donne à l'exécution :

```
1 2 3 4 5
```

l'instruction for

exemple de la boucle en C for (i=0; i<5 ; i++)

```
for i in range(5) :
    commandes
```

les instructions break, continue

L'instruction break permet d'arrêter une boucle avant sa fin. L'instruction continue est similaire, mais au lieu d'interrompre la boucle, elle revient au début de celle-ci.

```
for i in range(5):
    if i==3:
        break
    print i
```

affichera

```
0 1 2
```

tandis que

```
for i in range(5):
    if i==3:
        continue
    print i
```

affichera

```
0 1 2 4
```

Exception

```
try:
    # Code pouvant générer une exception
except MonException:
    # Code en cas d'exception
else:
    # Code en cas de non exception
#code dans tous les cas
```

Exemple : Exemple de gestion d'exception

Les chaînes de caractères

Une chaîne de caractères se déclare par l'utilisation de single quote (') ou de double quote (") :

```
chaîne='une chaîne de caractère'
chaîne2="une autre chaîne de caractères"
```

chaîne de caractère : Déclaration d'une chaîne de caractères

On peut aussi imbriquer les chaînes de caractères :

```
chaîne='Il lui a dit "Bonjour!"'
```

pour insérer des caractères spéciaux (saut de ligne, tabulation, etc), il est nécessaire de les encoder :

- Retour de ligne: \n
- Tabulation: \t
- backslash: \\
- etc...

Exemple :

```
print "une phrase\npar ligne"
```

Résultat :

```
une phrase
par ligne
```

Sinon on peut faire des chaînes de caractères entre triple quotes (' ' 'patati' ' ') ou triple guillemets ("" ""patata"" ""), dans ces cas, les guillemets et les quotes sont autorisés à l'intérieur, de même que les sauts de lignes.

Exemple :

```
print ""abc, ", \n q
go ""
```

Résultat :

```
abc, ",
q
go
```

de nombreuses méthodes sont associées à l'objet chaîne de caractère (str) :

Méthodes

les fonctions

```
str()
```

et

```
repr()
```

permettent de transformer un objet python quelconque en chaîne de caractères. Ces deux fonctions sont différentes :

```
str('chaîne')
```

retournera 'chaîne' tandis que

```
repr('chaîne')
```

retournera `""chaîne""`.

Autres méthodes

Les principale méthodes sont :

- `capitalize()` qui retourne la chaîne avec le premier caractère en majuscule
- `index(car)` qui retourne la position du caractère `car` ou déclenche une erreur (`IndexError`)
- `replace(old,new)` remplace la sous-chaîne `old` par `new`
- `split(car)` renvoie une liste en séparant la chaîne par le caractère `car`. Si `car` n'est pas donné, `split` sépare par les sauts de ligne et les espaces
- `strip()` enlève les espaces au début et à la fin
- `upper()` met en majuscule
- `lower()` en minuscule
- ... (voir le résultat de `help(str)` pour la liste complète)

Les listes

Les listes sont des tableaux.

L'encodage est comme suit :

```
>>MaListe=['a','b','c']
>>MaListe
['a','b','c']
```

Une sortie indicée donne :

```
>>MaListe[0]
'a'
```

Une séquence d'indice donne :

```
>>MaListe[0:2]
['a','b']
```

Tableau à trois lignes :

```
Tab = range(1, 5)
Tab[1] = u'ligne 1'
Tab[2] = u'ligne 2'
Tab[3] = u'ligne 3'

for ligne in range(1,4):
    print Tab[ligne]
```

Tableau à deux dimensions :

```

ligne = 3
colonne = 2
Tab = [[0] * (colonne+1) for _ in range(ligne+1)]
Tab[1][1] = u'1.1'
Tab[1][2] = u'1.2'
Tab[2][1] = u'2.1'
Tab[2][2] = u'2.2'
Tab[3][1] = u'3.1'
Tab[3][2] = u'3.2'

for l in range(1,ligne+1):
    for c in range(1,colonne+1):
        print Tab[l][c]

```

Les listes sont des objets

A ce titre, elles disposent de "méthodes".

L'exemple suivant qui fait appel à la méthode "append" de l'objet "liste" illustre ce concept.

Il s'agit de remplir la hotte du Père Noël.

```

#!/usr/bin/python
# -*- coding: iso8859-1 -*-

# on prépare une liste encore vide
cadeaux = []

# on va ajouter des éléments à la liste
unCadeau = ""

# la suite des éléments à introduire dans la liste est définie par l'utilisateur
# lorsqu'il a terminé, l'utilisateur indique le mot "fin" qui ne sera pas introduit dans la liste
while unCadeau <> "fin":
    unCadeau = raw_input ("Père Noël, je voudrais que tu m'apportes ")
    # si l'utilisateur n'a pas frappé "fin", le mot est ajouté à la liste
    if unCadeau <> "fin":
        cadeaux.append(unCadeau)

# finalement, on écrit la suite des éléments introduits dans la liste
for chaqueCadeau in cadeaux:
    print chaqueCadeau

```

Les dictionnaires

Le dictionnaire ou tableau associatif fonctionne dans la même logique.

Ajout d'une occurrence à la pile

```

>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}

```

Sortie d'un élément de la pile


```
>>> tel['jack']
4098
```

Suppression et ajout d'un élément

```
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
```

Sortie des clés

```
>>> tel.keys()
['guido', 'irv', 'jack']
>>> tel.has_key('guido')
1
```

Les tuples

Python propose un type de données appelé tuple, qui est assez semblable à une liste mais qui n'est pas modifiable.

Du point de vue de la syntaxe, un tuple est une collection d'éléments séparés par des virgules :

```
>>> tuple = 'a', 'b', 'c', 'd', 'e'
>>> print tuple
('a', 'b', 'c', 'd', 'e')
```

Bien que cela ne soit pas nécessaire, il est vivement conseillé de mettre le tuple en évidence en l'enfermant dans une paire de parenthèses, comme l'instruction `print` de Python le fait elle-même (il s'agit simplement d'améliorer la lisibilité du code, mais vous savez que c'est important) :

```
>>> tuple = ('a', 'b', 'c', 'd', 'e')
```

Les opérations que l'on peut effectuer sur des tuples sont syntaxiquement similaires à celles que l'on effectue sur les listes, si ce n'est que les tuples ne sont pas modifiables :

```
>>> print tuple[2:4]
('c', 'd')
>>> tuple[1:3] = ('x', 'y') ==> ***** erreur *****
>>> tuple = ('André',) + tuple[1:]
>>> print tuple
('André', 'b', 'c', 'd', 'e')
```

Remarquez qu'il faut toujours au moins une virgule pour définir un tuple (le dernier exemple ci-dessus utilise

un tuple contenant un seul élément : 'André'). Vous comprendrez l'utilité des tuples petit à petit. Signalons simplement ici qu'ils sont préférables aux listes partout où l'on veut être certain que les données transmises ne soient pas modifiées par erreur au sein d'un programme. En outre, les tuples sont moins « gourmands » en ressources système (ils occupent moins de place en mémoire).

Les chaînes de caractères sont des liste de caractères chaînés le plus souvent dans des tableaux à une dimension.

Chaîne simple

```
strings = "il etait une fois"

>>> print strings
'il etait une fois'
```

Liste de chaînes

```
strings = ["string1"]*5 + ["string2"]*3

>>> print strings
['string1', 'string1', 'string1', 'string1', 'string1', 'string2', 'string2', 'string2']

>>> print strings[0]
string1
```

Concaténation

append

str2 renvoie toutes les entrées de *strings* qui ne sont pas des *string2*, concaténées.

```
def str2():
    string_final = []
    for string in strings:
        if not "string2" in string:
            string_final.append(string)
    return "".join(string_final)

>>> str2()
'string1string1string1string1string1'
```

join

str3 retourne toutes les entrées de *strings* qui ne sont pas des *string2*, concaténées via *app*.

```
def str3():
    string_final = []
    app = string_final.append
    for string in strings:
        if not "string2" in string:
            app(string)
    return "".join(string_final)

>>> str3()
'string1string1string1string1string1'
```

Optimisation

str4 vérifie chaque entrée de *strings*, si elle se termine par *ing2* elle est ajoutée au résultat.

```
def str4():
    string_final = ""
    for string in strings:
        if string[:4] == "ing2":
            string_final += string
    return string_final

>>> str4()
```

str5 vérifie chaque entrée de *strings*, si elle se termine par *ing2* elle est ajoutée au résultat.

```
def str5():
    string_final = ""
    for string in strings:
        if string.startswith("ing2"):
            string_final += string
    return string_final

>>> str5()
```

On s'aperçoit que `str4()` est plus rapide que `str5()` car ne faisant pas appel à une fonction de plus haut niveau dans la boucle.

Définition

Une fonction est définie par le spécificateur `def` suivi du nom de la fonction et de ses paramètres.

```
def nomFonction(arg)
    return val
```

Utilisation

```
def fac (n):
```

```
f = 1
i = 1
while i <= n:
    f = f * i
    i = i + 1
return f # la valeur retournée
```

l'appel se fait par

```
fac(7) //> 720
```

Retour de valeur

```
>>> def f(x):
...     return x*2
```

Le return retourne la valeur

Passage d'argument

```
>>> def f(x,y):
...     return x*y
```

La signature est ici x et y en paramètre

Fonction lambda

Une fonction lambda est une fonction anonyme : elle n'est pas définie par *def*.

```
>>> def f(x):
...     return x*2
...
>>> f(3)
6
>>> g = lambda x: x*2 //1
>>> g(3)
6
>>> (lambda x: x*2)(3) //2
6
```

1 et 2 sont des fonctions lambdas.

Récupérer les arguments de la ligne de commande

la variable `sys.argv` contient les arguments de la ligne de commande, sous forme d'une liste dont le premier élément est le nom du script invoqué exemple :

si le script `truc.py` contient

```
#!/usr/bin/python
#-*- coding: utf-8 -*-
import sys
print u"Arguments : ", sys.argv
```

alors l'invocation :

```
$ python truc.py -a rien -n=nervures
```

produira la sortie

```
Arguments :  ['truc.py', '-a', 'rien', '-n=nervures']
```

si on veut récupérer le Argument 2

```
#!/usr/bin/python
#-*- coding: utf-8 -*-
import sys
print u"Arguments 2 : ", sys.argv[2]
```

produira la sortie

```
Arguments :  ['rien']
```

Définition

Un module peut être appelé depuis plusieurs programmes, il s'agit d'un fichier `.py` commençant par son identité (qui ne contient pas de point).

N'importe quel fichier `.py` peut donc être appelé depuis un autre comme un module^[1]. Il peut contenir :

- du script
- des fonctions
- des classes
- ...

Importations

Pour utiliser des fonctions de modules dans un programme, il faut au début du fichier importer ceux-ci.

Pour ce faire, utiliser la commande "import" :

```
import os
import codecs
```

Sur la même ligne :

```
import os, codecs
```

Ou encore en sélectionnant tous les éléments d'un fichier :

```
from wikipedia import *
```

A noter : cette dernière méthode est dangereuse, car des objets du module portant le même nom que des objets du programme peuvent s'écraser l'un l'autre.

TypeError

Si l'erreur suivante pose une colle lors de l'appel :

```
TypeError: 'module' object is not callable
```

Il suffit d'appeler le module sous la forme : NomFichier.NomFonction.

Import error

Si cette autre erreur arrive :

```
Import error: No module named urllib3.response
```

Il suffit de modifier le PYTHONPATH pour qu'il trouve le module mentionné, par exemple derrière une condition s'assurant que la machine qui exécute le script contient le répertoire du module :

```
import sys, socket
if socket.gethostname() == "MonUbuntu":
    sys.path.append(u'/usr/local/lib/python2.7/dist-packages/requests')
else:
    import requests
```

La page suivante traite du module *re*.

Références

- anglais <http://docs.python.org/2/tutorial/modules.html>

Syntaxe

Expressions rationnelles

Caractère	Type	explication
.	Point	n'importe quel caractère
[...]	classe de caractères	tous les caractères énumérés dans la classe
[^...]	classe complémentée	Tous les caractères sauf ceux énumérés
^	circonflexe	marque le début de la chaine, la ligne...
\$	dollar	marque la fin d'une chaine, ligne...
	barre verticale	alternative - ou reconnaît l'un ou l'autre
(...)	parenthèse	utilisée pour limiter la portée d'un masque ou de l'alternative
*	astérisque	0, 1 ou plusieurs occurrences
+	le plus	1 ou plusieurs occurrence
?	interrogation	0 ou 1 occurrence

Les expressions régulières en Python nécessitent le module *re*.

Recherche

La fonction `compile()` renvoie `None` si l'expression rationnelle n'est pas trouvée dans la chaine.

La fonction `search()` renvoie la position des chaines recherchées.

```
#!/usr/bin/env python
import re
chaine = "Test regex Python pour Wikibooks francophone."
if re.compile('Wikibooks').search(chaine):
    print "Position du mot Wikibooks : "
    print re.search(u'Wikibooks', chaine).start()
    # Affiche "23"
    print re.search(u'Wikibooks', chaine).end()
    # Affiche "32"
```

```
#!/usr/bin/env python
```

```
# Affiche tous les mots qui commencent par "Wiki"
import re
chaine = "Wikilivre regex Python pour Wikibooks francophone."
print (re.findall(r"Wiki\w+", chaine))
# Affiche ['Wikilivre', 'Wikibooks']
```

Remplacement

```
#!/usr/bin/env python
# Remplace tous les espaces par des underscores
import re
chaine = "Test regex Python pour Wikibooks francophone."
chaineTrie = re.sub(r' ', "_", chaine)
print chaineTrie
# Affiche "Test_regex_Python_pour_Wikibooks_francophone."
```

Pour remplacer certains éléments en conservant ceux placés entre parenthèses, il faut les désigner par \1, \2, \3...

```
#!/usr/bin/env python
# Ajoute des guillemets à tous les mots suivant "livre"
import re
chaine = "Test regex Python pour le livre Python de Wikibooks francophone."
chaineTrie = re.sub(r'(.*)livre (\w+)(.*)', r'\1livre "\2"\3', chaine)
print chaineTrie
# Affiche "Test regex Python pour le livre "Python" de Wikibooks francophone."
```

Remarque : si les paramètres (\1, \2...) sont remplacés par le symbole , vérifier que la chaîne regex est bien encodée avec r.

La programmation orienté objet

Le concept objet

Dans les environnements de développement informatique, il a fallu attendre assez longtemps pour voir émerger le concept de l'objet. Son apparition a permis la création de systèmes beaucoup plus complexes mais aussi très empreints de mimétisme. En effet, dans notre monde réel, nous sommes tous entourés d'objets qui ont très souvent deux critères d'appréciation.

Le critère descriptif

Ce premier est universel, il contient toutes les caractéristiques qui décrivent l'objet. Nous prendrons comme exemple un dé, si nous avons à le décrire, nous dirions qu'il possède 6 faces avec un chiffre allant de 1 à 6 sur chacune d'elles, que la somme de deux valeurs étant sur des faces opposées vaut 7, que chaque chiffre entre un et six y est repris une et une seule fois, qu'il est (souvent) de couleur rouge et de petite taille. Il serait possible de le décrire plus précisément, mais en réalité, indiquer qu'il est fait de bois, que les nombres sont représentés par une quantité de point qui leur est égal, qu'il dispose de coin arrondi... n'aurait pas été plus éloquent.

Le critère d'interaction

Le deuxième critère est celui d'interaction, il indique l'utilité de l'objet, les possibilités qu'il vous offre. Pour le dé nous pourrions indiquer que celui-ci peut rouler, mais ce n'est pas son rôle. De même, dans certaines circonstances, celui-ci peut vous servir de cale, mais ici encore, nous nous éloignons du sujet. Objectivement, le dé a pour rôle de donner un nombre compris entre son minimum et son maximum (inclus) au hasard. D'ailleurs, on peut ajouter que cela arrive après l'avoir lancé.

L'héritage et l'implémentation

Ici, nous avons décrit un objet, et il nous suffit de faire de même en informatique. Mais nous pourrions approfondir la description en indiquant aussi que le dé est en fait dérivé d'un objet de base : le cube. Ainsi nous pourrions dire que le dé :

- est un **cube**.
- est de couleur rouge.
- peut être lancé pour renvoyer un nombre compris entre 1 et 6 (le nombre de face qui le compose).

puis expliquer que le cube :

- est un volume géométrique à trois dimensions.
- est constitué de 6 **carrés**.

puis bien sûr qu'un **carré** :

- est une figure géométrique à deux dimensions.

Et nous pourrions continuer précisant le terme **dimension** mais dans notre cas ce n'est pas utile. Nous pouvons ainsi établir le schémas suivant : le dé hérite des caractéristiques du cube (c'est un cube). Mais on ne peut pas dire que le cube hérite des caractéristiques du carré. En effet, on indique bien qu'il est **constitué de** mais pas qu'il **est** et c'est la toute la différence, vous êtes constitué de deux bras musclés mais vous n'êtes pas deux bras musclés (sauf si vous êtes déménageur... *c'est une blague bien entendu, les déménageurs sont suffisamment allègre pour ne pas lancer un avis de recherche sur ma tête*) ! Nous dirons donc que :

- l'objet **cube** implémente l'objet **surface carré**
- l'objet **dé** hérite de l'objet **cube**

Classes et objets

Exemple de classe

```
class MaClasse:
    # définir ici les méthodes
```

Constructeur, destructeur

Classe avec constructeur

```
class Complexe:
    def __init__(self, r, i):
        self.reel = r
        self.img = i
```

Encapsulation des données

Héritage

Classe héritant d'une autre

```
class Triangle(FormeGeometrique):
    def __init__(self):
        # Création d'un triangle
```

Polymorphisme



Cette section est vide, pas assez détaillée ou incomplète.

Une des grandes forces du langage Python réside dans le nombre important de bibliothèques logicielles externes disponibles. Une bibliothèque est un ensemble de fonctions. Celles-ci sont regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire.

Celles-ci permettent de faire : du calcul numérique, du graphisme, de la programmation internet ou réseau, du formatage de texte, de la génération de documents...

Bibliothèques standards

La distribution standard de Python contient un certain nombre de bibliothèques qui ont été considérées comme suffisamment génériques pour intéresser la majorité des utilisateurs.

Leur utilisation est relativement bien expliquée dans la documentation de la distribution.

Les couches de présentation des applications (couche IHM avec wxPython, PyQt, PyKDE Tk, tkinter 3000, pyGTK, pybwidget, Pmw, TIX)

les couches controller des serveurs d'application Web (analyse HTML -htmllib, xmllib, urlParse, mimetools-
Serveur d'application : Zope - Django, Turbogears, CherryPy, Plone, GGI)

les couches Modele d'accès aux données (MySQL -MySQLdb- , Oracle -dcoracle-, MS SQL Server, PostgreSQL -psycopg-, FireBird -kinterbasdb- , SybODBC, GadFly, PySqlite, Zodb- BDD objet -)

la couche de persistance XML (4Suite, PySimpleXML, XmlSerializer, Jaxml) ou spécifique à Python (Cpickle, Shelve)

les couches d'accès au middleware ICE, COM/CORBA/.NET (win32, OmniORB, Ironpython) :
programmation orientée composant (pont vers des bibliothèques Fortran, C et C++)

les couches de communication standalone (port série : pySerial, port parallèle : pyParallel) , réseau (Twisted, urllib2, HTMLparser, ftplib], socket, poplib, rfc822, mailbox, mhlib, imaplib, smtplib, telnet, etc.)

les couches de frameWork bas niveau (ajout de capacité de script. exemple : Boost.Python)

Les couches multimédia : traitement d'image (PIL)

Les couches utilitaires :

- de gestion de l'heure (datetime, time)
- de compression (gzip)
- de codage/décodage de données binaires (hashlib -md5, sha- , base64, binHex, binascii)
- de structure de données (decimal, deque, array, dico, list, queue, heapq)
- de parallélisme (thread)
- d'expressions régulières (re)
- de différences (difflib)
- d'accès au dll ou.so (ctype)
- de manipulation de chaînes (string, str, StringIO)
- de parseur (standard - shlex, ConfigParser, email, parser, cmd - ou Tierce- pyBison, ples, pyparsing, ply, PyGgy, Yapps, pyLR)
- de calcul (math, numarray - tableaux multidimensionnaires - , cmath, random)
- de log (logging)

Le déploiement se fait soit en utilisant des modules d'installation standardisés (distutils), soit en générant un exécutable qui ne nécessite plus l'existence de l'interpréteur sur les machines cibles (Windows : py2exe, Cx_freeze; Unix : freeze)

Bibliothèques alternatives

- Un site internet recense la liste des bibliothèques utilisables avec ce langage: <http://www.vex.net/parnassus>
- Les bibliothèques les plus populaires et les plus utilisées (XML, interfaces graphiques...) bénéficient de pages dédiées sur le site principal du langage : <http://cheeseshop.python.org/pypi/%3Aaction=index>

Liste de Bibliothèques

Incomplète

- CWM : Modules de parseur pour le web semantique : <http://infomesh.net/2001/cwm/>
- epydoc : Utilisé pour générer la documentation : <http://epydoc.sourceforge.net/>
- guidata : Librairie graphique basée sur Qt dédiée à l'affichage de données : <http://pypi.python.org/pypi/guidata/>
- guiqwt : Librairie graphique basée sur Pyqwt dédiée à l'affichage de courbes : <http://packages.python.org/guiqwt/>
- SAGE : Logiciel d'algèbre et de géométrie (alternative à MATHematica, Maple ...) : géométrie, théorie des nombres, cryptographie, calcul numérique... : <http://modular.math.washington.edu/sage/>
- pyCLIPS : Module pour scripter le système expert CLIPS : <http://pyclips.sourceforge.net/>
- Pychinko: Implémentation de l'algorithme de Rete (pour le chaînage avant)
- pydot : Module pour scripter le générateur de graphique Graphviz : <http://dkbza.org/pydot.html>
- Pygame : Module de création de jeu 2D : <http://www.pygame.org/news.html>
- PySFML : Module de création de jeu 2D : <http://www.sfm1-dev.org/tutorials/1.6/start-python-fr.php>
- Pyglet : Module de création de jeu 2D utilisant l'openGL : <http://www.pyglet.org/>
- Cocos2d : Frameworks multiplateformes pour construire des jeux 2d, demos ou des applications interactives graphiques en openGL (nécessite Pyglet) : <http://cocos2d.org/>
- Karrigell (<http://karrigell.sourceforge.net>) [[archive](#)], CherryPy , TurboGears , Django , Web2py (<http://www.web2py.com>) [[archive](#)], Pylons (<http://www.pylonshq.com>) [[archive](#)] : Framework de développement web.
- Soya : Moteur 3D : <http://home.gna.org/oomadness/en/soya3d/index.html>

- Panda 3D : Moteur 3D : <http://www.panda3d.org/>
- PyOgre : Moteur 3D : <http://www.ogre3d.org/>
- Pmw, Pybwidget, Tkinter 3000, PyGTK : Interface graphique.
- pyinstaller : Création d'exécutable pour toute plateforme : <http://pyinstaller.python-hosting.com/>
- Py2exe : Créer un exécutable Windows pour vos scripts : <http://www.py2exe.org/> (version pour Python 2.7 (<http://www.py2exe.org/old/>) [[archive](#)])
- MySQLdb, Gadgetfly, Psycopg, Kinterbasdb, Buzhug : Base de données.
- Matplotlib : Bibliothèque de dessin de courbe 2D (très riche) : <http://matplotlib.sourceforge.net/>
- gnuplot-py : Bibliothèque pour s'interfacer avec gnuplot : <http://gnuplot-py.sourceforge.net/>
- PyNGL : Bibliothèque pour créer des graphes 2D : <http://www.pyngl.ucar.edu/index.shtml>
- scipy et numpy : Framework pour le calcul scientifique : Interpolation, intégration (ODE integrators), algèbre linéaire (LAPACK), Interpolation, systèmes dynamiques (PyDSTool) : <http://www.scipy.org/>
- PyIMSL Studio : Framework pour le calcul scientifique s'appuyant sur les bibliothèques mathématiques et statistiques IMSL : <http://sites.google.com/site/roguewavesoftwarefrance/produits/PyIMSL-Studio>
- PIL : Manipulation et traitement d'image : <http://www.pythonware.com/products/pil/>
- SVGdraw : Création d'image au format SVG (Scalable Vector Graphics) : <http://www2.sfk.nl/svg>
- pygsl : Interface vers GNU scientific library (gsl): vecteur, matrice, transformation de fourrier, recuit simulé, algèbre linéaire... : <http://pygsl.sourceforge.net/>
- CGAL: CGAL-Python bindings pour la CGAL library (Computational Geometry Algorithms Library) : <http://cgal-python.gforge.inria.fr/>
- GMPY : General Multiprecision PYthon : Interface vers la bibliothèque de précision arithmétique GNU GMP : <http://gmpy.sourceforge.net/>
- pyrorobotics : Environnement pour l'étude de la robotique et l'intelligence artificielle. Réseaux de neurones : <http://www.pyrorobotics.org/>
- pybel : Interface pour la bibliothèque Open source de CHIMIE Open Babel.
- FANN : Fast Artificial Neural Network Library : binding Python pour FANN : <http://leenissen.dk/fann/index.php>
- Maximum Entropy Modeling Toolkit : Framework qui met en oeuvre le principe de l'entropie maximum : http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html
- Orange : Technique d'exploration de données, data mining : <http://www.ailab.si/orange>
- MayaVi2 : Visualisation des données scientifiques en 2D/3D : <https://svn.enthought.com/enthought/wiki/MayaVi>
- pySerial : Manipulation des ports séries, non intégrés par défaut, bien que très souvent utilisés dans le monde industriel : <http://pyserial.sourceforge.net/>
- pyParallel : Accès aux ports parallèles : <http://pyserial.sourceforge.net/pyparallel.html>
- pyMPI : Calcul parallèle : <http://pympi.sourceforge.net/index.html>
- PyPar : Calcul parallèle : <http://datamining.anu.edu.au/~ole/pypar/>
- PyVISA : Contrôle des ports GPIB, RS232, and USB : <http://pyvisa.sourceforge.net/>
- PyUSB : Manipulation du port USB : <http://pyusb.berlios.de/>
- pyro : Middleware Python Remote Objects : <http://pyro.sourceforge.net/>
- pyX : Python graphics package - Analyse de donnée : <http://pyx.sourceforge.net/>
- simPy : Simulation de systèmes dynamiques à file d'attente : <http://simpy.sourceforge.net/>
- Twisted : Pluggable, asynchronous protocols (TCP, UDP, SSL/TLS, multicast, Unix sockets, HTTP, NNTP, IMAP, SSH, IRC, FTP) : <http://twistedmatrix.com/trac/>
- SCons : Alternative puissante à make (next-generation build tool) : <http://www.scons.org/>
- VPython : Simulation 3D basée sur OpenGL : <http://www.vpython.org/>
- directpython: Binding vers Direct X : <http://directpython.sourceforge.net/>
- pymedia : Module for wav, mp3, ogg, avi, divx, dvd, cdda etc. files manipulations : <http://pymedia.org/>
- wxPython : Bibliothèque d'accès à un toolkit très puissant (en particulier pour les interfaces graphiques) : <http://www.wxpython.org/>
- PyML : Python machine learning package : Framework pour l'apprentissage par les machines (data mining ...) : <http://pymml.sourceforge.net/>

- Zope : Serveur d'application web orienté objet et base de données Objet : <http://www.zope.org/>
 - GENA : Algorithme génétique en Python : <http://www.madiku.org/ylrt3i0sfy/?p=1291>
-

L'interface graphique pour Python

Comme dans tout langage de programmation, l'exécution du code ne se voit pas par l'utilisateur !

Jusqu'à maintenant, la seule relation entre le programme une fois lancé et l'utilisateur était le terminal :

Un *print* par-ci, pour donner des informations à l'utilisateur, uniquement sous forme d'une chaîne de caractères.

Un *input* par-là, afin que l'utilisateur puisse envoyer des données, toujours sous forme d'une chaîne de caractères, au programme.

Dorénavant, ayant des acquis sur la programmation dans ce langage, il va vous être enfin possible de donner du style à vos programme en créant une véritable Interface Homme-Machine (IHM) !

Pour cela, python intègre déjà avec son interpréteur : Tkinter, qui est une bibliothèque graphique libre. Créer vos Interfaces Homme-Machine avec cette bibliothèque permettra à l'utilisateur de n'avoir aucune bibliothèque à télécharger en plus de votre code. Il sera donc très portable !

Sinon, dans les bibliothèques graphiques libres, les principaux modules sont :

- Tkinter pour Tk
- wxPython pour wxWidgets
- PyGTK pour GTK+
- PyQt pour Qt.

Inutile d'en dire davantage ; les liens Wikipédia sont très bien documentés. Sachez qu'il n'y a pas une solution unique ; ces bibliothèques ont toutes leurs caractéristiques ; tout avis ici ne serait que subjectif ; par conséquent, la seule chose qu'il reste à dire est : essayez-les, et choisissez celle que vous voudrez... L'esprit du libre, c'est aussi d'avoir le choix !

Créer des interfaces python avec Tkinter

Que faut il posséder d'abord ?

Tkinter (pour Tool kit interface) est une boîte à outils d'interface graphique pour Python. L'interface Tkinter s'installe avec Python. Il suffit donc juste d'installer Python 2.3 ou 2.4 ou supérieur pour pouvoir utiliser Tkinter^(Référence nécessaire).

Ensuite il vous suffit d'importer la bibliothèque dans votre programme : `from Tkinter import *`

Principes de base

Créez un fichier texte dans le dossier python 2.4 ou python 2.3.

Donnez lui le nom que vous voulez, mais pour que votre fichier puisse être interprété par python, il doit porter l'extension .py ou .pyw

Créez donc un fichier du type : monfichier.py (dans ce cas, la console s'affichera, ce qui peut être pratique pour suivre le fonctionnement du programme)

ou monfichier.pyw (dans ce cas la console ne s'ouvrira pas : c'est ce type d'extension qu'il faut utiliser pour la version définitive du programme).

Pour modifier le programme :

- clic droit sur le fichier
- ouvrir avec un logiciel d'édition de texte pur.

Quelques règles d'écriture

Les espaces n'ont pas d'incidence sur le fonctionnement Les commentaires, précédés de #, permettent de donner des indications et des repères dans le programme, mais n'ont aucun effet sur le programme.

Créer une fenêtre pour l'application

Propriétés et méthodes de l'objet fenêtre

le programme est le suivant :

```
from Tkinter import * # le programme va aller chercher toutes les fonctions de la librairie Tkinter
Fenetre= Tk()         # vous pouvez choisir le nom que vous voulez pour votre fenêtre
Fenetre.mainloop()    # lance la boucle principale
```

Qu'est-ce qu'un widget ?

widget : contraction de l'anglais *windows gadget* (gadget fenêtre). Les widgets sont tous les objets graphiques que l'on peut insérer dans une interface (fenêtre). Les principaux sont :

- Les boutons : Button (pour commander une action)
- Les labels : Label (pour insérer un texte)
- Les zones de saisie : Entry (pour permettre l'entrée d'une donnée)
- Les canevas : Canvas (pour insérer des dessins)

Chaque widget a des propriétés et des méthodes qui permettent de régler son apparence et les interactions avec l'utilisateur.

Le widget Button

Chaque widget (objet d'interface) doit être créé puis placé dans la fenêtre

```
#!/usr/bin/python          # Emplacement de l'interpréteur Python (sous Linux)
# -*- coding: utf-8 -*-   # Définition l'encodage des caractères
from Tkinter import *     # le programme va aller chercher toutes les fonctions de la librairie Tkinter
```

```
Fenetre= Tk() # création de la fenêtre, avec un nom de votre choix
bouton=Button(Fenetre, text="quitter", command=Fenetre.destroy) # Bouton qui détruit la fenêtre
bouton.pack()      # insère le bouton dans la fenêtre
Fenetre.mainloop() # lance la boucle principale
```

- La commande `Fenetre.destroy()` est une méthode de destruction qui s'applique à l'objet `fen`. La pression du bouton a donc pour conséquence la fermeture de la fenêtre `Fenetre`

propriétés et méthodes de l'objet "bouton"

- Ce qui sera affiché sur votre bouton est contenu dans la propriété `"text"`, passée ici en paramètre de l'objet `"bouton"`.
- La procédure `"command"` permet une action lorsqu'on clique sur ce bouton. Cette procédure peut également être choisie parmi les fonctions définies dans le programme.

Le widget Label

L'incontournable "Hello world"

```
#!/usr/bin/python          # Emplacement de l'interpréteur Python (sous Linux)
# -*- coding: utf-8 -*-   # Définition l'encodage des caractères
from Tkinter import *
Fenetre=Tk()
texte=Label(Fenetre, text="Hello World",fg='black') # Création du texte "Hello World" de couleur noire
texte.pack() # Insère le texte dans la fenêtre
Fenetre.mainloop()
```

Propriétés et méthodes de l'objet label

- `"fg"` contient la couleur du texte (en anglais)
- `"bg"` contient la couleur de fond du texte (en anglais)

Le widget Entry

Propriétés et méthodes de l'objet Entry

L'objet `Entry()` est une zone de saisie de texte que l'on crée de la manière suivante:

```
#!/usr/bin/python          # Emplacement de l'interpréteur Python (sous Linux)
# -*- coding: utf-8 -*-   # Définition l'encodage des caractères
from Tkinter import * #On importe l'ensemble du module Tkinter
Fenetre = Tk()
Entree = Entry(Fenetre)    # On définit l'objet Entry qui porte le nom Entree
Entree.pack()              # On place "Entree"
Fenetre.mainloop()        # On lance la boucle du programme
```

Vous pouvez l'utiliser dans des situations plus complexes comme, par exemple, un formulaire que je vous laisserai examiner:

```
#!/usr/bin/python          # Emplacement de l'interpréteur Python (sous Linux)
# -*- coding: utf-8 -*-   # Définition l'encodage des caractères
```

```

from Tkinter import *

def repondre():
    affichage['text'] = reponse.get()      # lecture du contenu du widget "reponse"

Fenetre = Tk()
Fenetre.title('Mon nom')

nom = Label(Fenetre, text = 'Votre nom :')
reponse = Entry(Fenetre)
valeur = Button(Fenetre, text = ' Valider', command=repondre)
affichage = Label(Fenetre, width=30)
votre_nom=Label(Fenetre, text='Votre nom est :')
nom.pack()
reponse.pack()
valeur.pack()
votre_nom.pack()
affichage.pack()

Fenetre.mainloop()

```

Le widget Canvas

Le widget Canvas (canevas, en français) est une zone de dessin rectangulaire.

Notons que l'angle haut gauche du canevas est l'origine des coordonnées (x,y)=(0,0).

Un exemple d'utilisation :

```

#!/usr/bin/python          # Emplacement de l'interpréteur Python (sous Linux)
# -*- coding: utf-8 -*-   # Définition l'encodage des caractères

from Tkinter import *

racine= Tk()

zone_dessin = Canvas(racine, width=500, height=500) #Définit les dimensions du canevas
zone_dessin.pack() #Affiche le canevas
zone_dessin.create_line(0,0,500,500) #Dessine une ligne en diagonale
zone_dessin.create_rectangle(100,100,200,200) #dessine un rectangle

bouton_sortir = Button(racine,text="Sortir",command=racine.destroy)
bouton_sortir.pack()

racine.mainloop()

```

Quelques propriétés de l'objet Canvas

Les propriétés sont définies en paramètre lors de la construction de l'objet

- height : Hauteur Y du canvas
- width : Largeur X du canvas
- bg : Couleur de fond du canvas
- bd : Taille en pixels du bord du canvas (2 par défaut)
- relief : Style de la bordure (flat (par défaut),raised,sunken,groove,ridge)
- ...

Quelques méthodes du widget Canvas

- .create_arc(): Pour créer un arc de cercle
- .create_bitmap(): Image bitmap

- `.create_image()`: Image graphique
- `.create_line()`: Pour créer une ligne
- `.create_oval()`: Pour créer un cercle ou une ellipse
- `.create_polygon()`: Pour créer un polygone
- `.create_rectangle()`: Pour créer un rectangle
- `.create_text()`: Texte
- `.create_window()`: Une fenêtre rectangulaire

Exemple supplémentaire

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from Tkinter import *

Fenetre=Tk()      #La fonction Tk() du module Tkinter permet de créer une fenêtre qui se nomme Fenetre
Fenetre.title("Mon programme avec Tkinter") # Donne un titre à la fenêtre (par défaut c'est Tk)

# Dans Fenetre nous allons créer un objet type Canvas qui se nomme zone_dessin
# Nous donnons des valeurs aux propriétés "width", "height", "bg", "bd", "relief"
zone_dessin = Canvas(Fenetre,width=500,height=500,
                      bg='yellow',bd=8,relief="ridge")
zone_dessin.pack() #Affiche le Canvas

#Nous allons maintenant utiliser quelques méthodes du widget "zone_dessin"
zone_dessin.create_line(0,0,500,500,fill='red',width=4) # Dessine une ligne
zone_dessin.create_line(0,500,500,0,fill='red',width=4) # Dessine une ligne
zone_dessin.create_rectangle(150,150,350,350) # Dessine un rectangle
zone_dessin.create_oval(150,150,350,350,fill='white',width=4) # Dessine un cercle

# boutons_sortir est un widget de type "Button"
# dont nous définissons les propriétés "text" et "command")
bouton_sortir= Button(Fenetre,text="Sortir",command=Fenetre.destroy)
# la commande "destroy" appliquée à la fenêtre détruit l'objet "Fenetre" et clôture le programme
bouton_sortir.pack()

Fenetre.mainloop() # Lancement de la boucle du programme, en attente d'événements (clavier, souris,...)
```

Images .gif

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from Tkinter import *
Fenetre=Tk()
photo=PhotoImage(file="Wikibooks.gif")
labl = Label(Fenetre, image=photo)
labl.pack()
Fenetre.mainloop()
```

Voir aussi

- PMW (<https://sourceforge.net/projects/pmw/files/>) [\[archive\]](#) (Python megawidgets) pour ajouter des menus déroulants, cases à cocher et autres boîtes de dialogues.
- PIL (<http://www.pythonware.com/products/pil/>) [\[archive\]](#) (Python Imaging Library) pour incruster des images.
- py2.exe (<https://sourceforge.net/projects/py2exe/files/>) [\[archive\]](#) : pour créer des exécutables (limité à Python 2.6).

- tkRAD: Tkinter XML widget builder (<https://github.com/tarball69/tkRAD/wiki/Accueil>) [\[archive\]](#) : pour générer automatiquement des widgets Tkinter à partir d'un fichier source XML.
- anglais Site officiel (<http://docs.python.org/library/tkinter.html>) [\[archive\]](#)

Python contient des modules pour bases de données, comme MySQL, PostgreSQL, SQLite, IBM Informix Dynamic Server et Berkeley DB.

Par exemple pour installer le premier :

1. Sur Unix, taper : *apt-get install python-mysqldb*.
2. Sur Windows, télécharger en cliquant ici (<http://sourceforge.net/projects/mysql-python/>) [\[archive\]](#) ou là (<http://stackoverflow.com/questions/316484/problem-compiling-mysqldb-for-python-2-6-on-win32/319007#319007>) [\[archive\]](#).

MySQL

L'exemple ci-dessous permet d'exécuter une requête SQL de sélection :

```

1 import MySQLdb
2 db = MySQLdb.connect("machine", "dbuser", "password", "dbname")
3 cursor = db.cursor()
4 query = """SELECT * FROM sampletable"""
5 lines = cursor.execute(query)
6 data = cursor.fetchall()
7 db.close()

```

1. On ouvre l'accès aux fonctions du Module `MySQLdb`.
2. On tente d'établir une connexion à la base de données nommée (si cela ne fonctionne pas, essayer de pinger le nom de machine mentionné pour diagnostiquer un problème réseau).
3. La ligne trois définit l'objet "cursor" qui va servir d'interface avec la base de données.
4. On prépare ensuite la commande en langage SQL (ce qui pourrait aussi être défini en tout début du programme).
5. On exécute cette requête dans la base.
6. On formate les données brutes du résultat
7. On clos la connexion.

Remarque : quand il y a beaucoup de lignes, il est préférable d'utiliser `row = cursor.fetchone()` pour une meilleure visibilité :

```

1 import MySQLdb
2 db = MySQLdb.connect("machine", "dbuser", "password", "dbname")
3 cursor = db.cursor()
4 query = """SELECT * FROM sampletable"""
5 lines = cursor.execute(query)
6 while True:
7     row = cursor.fetchone()
8     if row == None: break
9 db.close()

```

Le résultat du `fetchone()` est de type `Tuple`.

Par ailleurs, la connexion à la base (en ligne 2) peut être enregistrée dans un fichier de configuration, celle-ci devient alors :

```
import MySQLdb
db = MySQLdb.connect(read_default_file=~/.my.cnf)
...
```

Postgres

```
import psycopg2
conn = psycopg2.connect("dbname=test")
cursor = conn.cursor()
cursor.execute("select * from test");
for i in cursor.next():
    print i
conn.close()
```

Références



Tout ou partie de cette leçon est issu de la traduction d'une page sous licence GFDL « anglais *Python Programming/Databases* » .

Consultez l'historique de la page originale ([//en.wikibooks.org/wiki/Python_Programming/Databases?action=history](http://en.wikibooks.org/wiki/Python_Programming/Databases?action=history)) [\[archive\]](#) pour connaître la liste de ses auteurs.



Liens externes

- anglais documentation SQLite (<http://docs.python.org/library/sqlite3.html>) [\[archive\]](#)
- anglais Psycopg2 (module PostgreSQL) (<http://initd.org/>) [\[archive\]](#)
- anglais module MySQL (<http://sourceforge.net/projects/mysql-python/>) [\[archive\]](#)
- anglais module IBM Informix Dynamic Server (<http://informixdb.sourceforge.net/>) [\[archive\]](#)

La Common Gateway Interface permet d’exécuter des programmes en Python sur un serveur HTTP.

Installation

Par défaut, lire un fichier .py en HTTP renvoie son contenu. Pour que le serveur compile et exécute de le code source, il faut que ce dernier soit placé dans un répertoire contenant un fichier nommé *.htaccess*, avec les lignes :

```
AddHandler cgi-script .py
```

Options +ExecCGI

Références

- anglais <http://docs.python.org/howto/webrowsers.html>
- <http://www.siteduzero.com/tutoriel-3-39020-apercu-de-la-cgi-avec-python.html>

Récupérer une page Web en python

Python intègre le module `httplib` (<http://docs.python.org/lib/module-httplib.html>) [\[archive\]](#) qui permet d'émettre et de recevoir des requêtes HTTP.

Afficher une page Web

Le code suivant ([src \(http://www.faqts.com/knowledge_base/view.phtml/aid/5152/fid/245\)](http://www.faqts.com/knowledge_base/view.phtml/aid/5152/fid/245) [\[archive\]](#)) permet de récupérer, à l'aide d'une requête HTTP, une page Web et affiche son code source à l'écran.

Exemple d'appel HTTP GET

```
# On utilise le module httplib
import httplib

# Connexion au proxy
# (si vous n'êtes pas derrière un proxy, alors mettre directement 'fr.wikibooks.org')
conn = httplib.HTTP('proxy:3128')

# Requête GET
# (si vous n'êtes pas derrière un proxy, alors mettre directement
# '/w/index.php?title=Programmation_Python_Le_r%C3%A9seau&action=edit'
conn.putrequest('GET', 'http://fr.wikibooks.org/w/index.php?title=Programmation_Python_Le_:'

conn.putheader('Accept', 'text/html')
conn.putheader('Accept', 'text/plain')

# Décommenter les 2 lignes suivantes si votre proxy nécessite une authentification
# auth = "Basic " + "username:password".encode('base64')
# h1.putheader('Proxy-Authorization', auth)

conn.endheaders()

# Récupération de la réponse
errcode, errmsg, headers = conn.getreply()
```

```
# Affichage d'éventuelles erreurs
print errcode
print errmsg
print headers

# Affichage de la réponse ligne après ligne
f=conn.getfile()
for line in f:
    print line

# fin de la connexion
conn.close()
```

Pygame est une bibliothèque écrite en **C**, pour **Python**.

Pygame se base sur la bibliothèque SDL, librairie C conçue pour la création de jeux vidéo en 2 dimensions.

Pygame gère l' affichage d' images, la gestion des événements, les sprites, ...

Site officiel : www.pygame.org (<http://www.pygame.org/>) [\[archive\]](#)

Initialisation et importation

L' importation se fait à l' aide de la commande

```
import pygame
from pygame.locals import *
```

La première ligne est la seule réellement nécessaire : mais la documentation officielle conseille d' ajouter la deuxième.

L' initialisation se fait à l' aide de la commande

```
pygame.init()
```

Il existe dans Python une bibliothèque permettant de manipuler les fichiers XML. Elle implémente la manière SAX (Simple API for XML) et DOM (Document Object Model).

Voyons comment manipuler simplement les fichier XML grâce à la méthode SAX.

La méthode SAX

Cette méthode est basée sur les évènements : une fonction est appelée lorsque l'on ouvre une balise, une autre lorsque le programme rencontre du texte, et une autre encore lorsque la balise se ferme.

Ces évènements sont définis dans une classe nommée **interface**. Cette classe doit dériver de ContentHandler,

contenu dans le module *xml.sax*, et peut implémenter les fonctions suivantes :

- *startElement()* est la fonction appelée lors de l'ouverture d'une balise. Les deux arguments sont le nom et un dictionnaire contenant les attributs.
- *endElement()* est la fonction appelée lors de la fermeture d'une balise. La fonction prend le nom de la balise en argument.
- *characters()* est appelée lors que le parseur rencontre un caractère en dehors d'une balise. Ce caractère est passé en paramètre.

Une fois cette classe faite, il faut créer un parseur. Cela est fait grâce à la fonction **make_parser()**, située elle aussi dans le module *xml.sax*. On lui associe ensuite une instance de l'interface avec la méthode **setContentHandler()**, et on lance l'analyse du fichier grâce à la méthode **parse**, qui prend le nom du fichier en argument.

Voici donc le code final :

```
from xml.sax.handler import ContentHandler
import xml.sax
import sys

class compteurElements(ContentHandler):
    def __init__(self):
        self.elem={}
    def startElement(self,name,attr):
        if not self.elem.has_key(name):
            self.elem[name] = 0
        self.elem[name] += 1
    def characters(self,ch):
        print ch
    def endElement (self, name):
        print name + ":" + str(self.elem[name])

parser = xml.sax.make_parser()
handler = compteurElements()
parser.setContentHandler(handler)
parser.parse(sys.argv[1])
```

Exemple 1 : Un compteur d'éléments bavard

La méthode DOM



Cette section est vide, pas assez détaillée ou incomplète.

Cette partie du livre *Programmation Python* présente les ouvrages et les sites Web ayant permis sa rédaction et permettant au lecteur de poursuivre son étude du sujet.

Bibliographie

- *Python - précis & concis* par Mark Lutz, aux éditions O'Reilly. ISBN 2-84177-360-4
- *Apprendre à programmer avec Python*, par Gérard Swinnen, aux éditions O'Reilly. ISBN 2-84177-299-3
- *Programmation Python*, par Tarek Ziadé, aux éditions Eyrolles. ISBN 2-212-11677-2

Livres en ligne

- français Apprendre à programmer en Python (<http://www.ulg.ac.be/cifen/inforef/swi/python.htm>) [\[archive\]](#)
- français Plongez au cœur de Python (<http://diveintopython.adrahon.org/>) [\[archive\]](#)
- anglais How to Think Like a (Python) Programmer (<http://www.greenteapress.com/thinkpython/>) [\[archive\]](#) (licence GFDL)

Site Web

- anglais Site officiel de Python (<http://www.python.org>) [\[archive\]](#) et son wiki (<http://wiki.python.org/moin/>) [\[archive\]](#)
- français Association francophone de Python (<http://www.afpy.org/>) [\[archive\]](#)
- français Site collaboratif français sur Python (<http://wikipython.flibuste.net/moin.py/FrontPage>) [\[archive\]](#)
- français Recueil de liens commentés (<http://www.python-eggs.org/links.html>) [\[archive\]](#)
- français Tutoriel python (<http://lfe.developpez.com/tutoriel/python/>) [\[archive\]](#)
- français Python est mon ami (http://ludovic.pinelli.free.fr/Python/Python_ami.html) [\[archive\]](#)

Liste des tableaux utilisés dans le livre Programmation Python:

- Tableau des mots réservés
- Tableau des types
- Tableau des valeurs False



GFDL

Vous avez la permission de copier, distribuer et/ou modifier ce document selon les termes de la **licence de documentation libre GNU**, version 1.2 ou plus récente publiée par la Free Software Foundation ; sans sections inaltérables, sans texte de première page de couverture et sans texte de dernière page de couverture.

Récupérée de « http://fr.wikibooks.org/w/index.php?title=Programmation_Python/Version_imprimable&oldid=440115 »

Dernière modification de cette page le 9 février 2014 à 20:17.

Les textes sont disponibles sous licence Creative Commons attribution partage à l'identique ; d'autres termes peuvent s'appliquer.

Voyez les termes d'utilisation pour plus de détails.

Développeurs